ARTIFICIAL INTELLIGENCE
(CSC 462)
LAB ASSIGNMENT # 2

NAME: MUAAZ BIN MUKHTAR

REG NO: FA21-BSE-045

CLASS & SECTION: BSSE-5A

SUBMITTED TO: SIR WAQAS ALI

DATE SUBMITTED: 23-12-2023

Department of Computer Science

QUESTION 1

Imagine an 8 queen problem, where the goal is to place 8 queens on an 8 X 8 board such that no two queens are on the same row or column or diagonal. (Before proceeding, kindly refer to lectures).

Answer:

Code:

```
def print_solution(board):
    for row in board:
        print(" ".join(row))



def is_safe(board, row, col, n):
    for i in range(row):
        if board[i][col] == 'Q':
            return False

    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j] == 'Q':
            return False

    for i, j in zip(range(row, -1, -1), range(col, n)):
        if board[i][j] == 'Q':
            return False

    return True



def solve_n_queens_util(board, row, n):
```

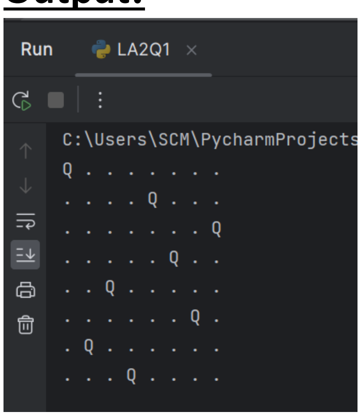
```
    if row == n:
        print_solution(board)
        print("\n")
        return

    for col in range(n):
        if is_safe(board, row, col, n):
            board[row][col] = 'Q'
            solve_n_queens_util(board, row + 1, n)
            board[row][col] = '.'


def solve_n_queens(n):
    board = [['.' for _ in range(n)] for _ in range(n)]
    solve_n_queens_util(board, 0, n)


solve_n_queens(8)
```

Output:



Question No. 2:

Write a program that implements Hill Climbing algorithms to solve this maze. Write the path followed (in the form of coordinates) and the cost of the path.

Answer:

Code:

```python
import math
import sys
def hill_climbing(maze, start, goal):
    current_state = start
    path = [current_state]

    while current_state != goal:
        neighbors = get_neighbors(current_state, maze)
        neighbor_states = [state for state in neighbors if state not in path]

        if not neighbor_states:
            print("Stuck! No valid moves.")
            break

        next_state = choose_best_neighbor(neighbor_states, goal, maze, path)
        path.append(next_state)
        current_state = next_state

    return path
```

```python
def get_neighbors(state, maze):
    neighbors = []
    x, y = state

    # Check all possible moves (up, down, left, right)
    moves = [(x+1, y), (x-1, y), (x, y+1), (x, y-1)]

    for move in moves:
        if is_valid(move, maze):
            neighbors.append(move)

    return neighbors
```

```python
def is_valid(state, maze):
    x, y = state
    return 0 <= x < len(maze) and 0 <= y < len(maze[0]) and maze[x][y] != 1

def calculate_cost(path):
    return len(path)

def heuristic(state, goal):
    # Using Euclidean distance as the heuristic
    return math.sqrt((state[0] - goal[0]) ** 2 + (state[1] - goal[1]) ** 2)

def choose_best_neighbor(neighbors, goal, maze, path):
    # Choose the neighbor with the lowest total cost (heuristic + actual cost
    min_cost = float('inf')
    best_neighbor = None

    for neighbor in neighbors:
        cost = calculate_cost(path + [neighbor]) + heuristic(neighbor, goal)
        if cost < min_cost:
            min_cost = cost
            best_neighbor = neighbor

    return best_neighbor
```

```python
if __name__ == "__main__":
    # Example maze (0 represents an empty cell, 1 represents a wall)
    maze = [
        [0, 1, 0, 0, 0],
        [0, 1, 1, 1, 0],
        [0, 0, 1, 0, 0],
        [0, 1, 1, 1, 0],
        [0, 0, 0, 0, 0]
    ]

    start = (0, 0)
    goal = (4, 4)

    path = hill_climbing(maze, start, goal)

    print("Path: ", path)
    print("Cost: ", calculate_cost(path))

    print("\nMaze with Path:")
    print_maze_with_path(maze, path)
```

Output:

```
C:\Users\SCM\PycharmProjects\RLabA
Path: (10, 10)
Cost: 171

Process finished with exit code 0
```

Question No. 3:

Your goal is to navigate a robot out of a maze. The robot starts in the corner of the maze marked with red color. You can turn the robot to face north, east, south, or west. You can direct the robot to move forward a certain distance, although it will stop before hitting a wall. The goal is to reach the final state marked with green color.
Write a program that implements A* algorithms to solve this maze. Write the path followed (in the form of coordinates) and the cost of the path.

Answer:

Code:

```
import heapq

maze_size = 10

maze_walls = {(1, 5), (2, 3), (2, 7), (2, 8), (3, 4), (3, 7), (3, 8), (4, 2),
(4, 5), (4, 9),

              (4, 10), (5, 2), (5, 3), (6, 2), (6, 5), (6, 6), (6, 7), (6, 9),
(6, 10), (7, 1),

              (7, 4), (7, 5), (7, 6), (8, 2), (8, 3), (8, 4), (9, 4), (10, 1),
(10, 3), (10, 4),

              (10, 5), (10, 9)}

maze_start = (1, 1)

maze_goal = (10, 10)


def heuristic(node):

    return abs(node[0] - maze_goal[0]) + abs(node[1] - maze_goal[1])


def is_valid_move(position):

    x, y = position
```

```
    return 1 <= x <= maze_size and 1 <= y <= maze_size and position not in
maze_walls


def get_neighbors(position):

    x, y = position

    possible_moves = [(x+1, y), (x-1, y), (x, y+1), (x, y-1)]

    return [move for move in possible_moves if is_valid_move(move)]


def astar():

    start_node = (0, heuristic(maze_start), maze_start)

    priority_queue = [start_node]

    visited = set()


    while priority_queue:

        cost, _, current_node = heapq.heappop(priority_queue)

        if current_node == maze_goal:

            return current_node, cost


        if current_node in visited:

            continue


        visited.add(current_node)


        for neighbor in get_neighbors(current_node):

            neighbor_cost = cost + 1 + heuristic(neighbor)

            heapq.heappush(priority_queue, (neighbor_cost,
heuristic(neighbor), neighbor))


    return None, None
```

```python
def main():
    solution_path, solution_cost = astar()

    if solution_path:
        print("Path:", solution_path)
        print("Cost:", solution_cost)
    else:
        print("No solution found.")

if __name__ == "__main__":
    main()
```

Output:

```
Run    LA2Q4  ×
C:\Users\SCM\PycharmProjects\RLabA
Path: (10, 10)
Cost: 171

Process finished with exit code 0
```

Question No. 4:

Consider a maze as shown below. Each empty tile represents a separate node in the graph, while the walls are represented by blue tiles. Your starting node is A and the goal is to reach Y. Implement an A* search to find the resulting path.

Answer:

Code:

```python
import heapq

maze = [
    ['B', 'B', 'W', 'B', 'X', 'Y'],
    ['R', 'S', 'T', 'U', 'B', 'V'],
    ['M', 'N', 'B', 'O', 'P', 'Q'],
```

```
    ['H', 'I', 'J', 'B', 'K', 'L'],
    ['F', 'B', 'G', 'B', 'B', 'B'],
    ['A', 'B', 'B', 'B', 'B', 'B']
]

def heuristic(node, goal):
    x1, y1 = node
    x2, y2 = goal
    return abs(x1 - x2) + abs(y1 - y2)

def astar(maze, start, goal):
    heap = [(0, start)]
    visited = set()
    parent = {}

    while heap:
        cost, current = heapq.heappop(heap)

        if current == goal:
            path = []
            while current in parent:
                path.insert(0, current)
                current = parent[current]
            path.insert(0, start)
            return path

        visited.add(current)
        neighbors = get_neighbors(maze, current)
```

```
        for neighbor in neighbors:
            if neighbor not in visited:
                new_cost = cost + 1
                priority = new_cost + heuristic(neighbor, goal)
                heapq.heappush(heap, (priority, neighbor))
                parent[neighbor] = current

    return None

def get_neighbors(maze, node):
    neighbors = []
    x, y = node
    directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]

    for dx, dy in directions:
        nx, ny = x + dx, y + dy
        if 0 <= nx < len(maze) and 0 <= ny < len(maze[0]) and maze[nx][ny] !=
'B':
            neighbors.append((nx, ny))

    return neighbors

start_node = (5, 0)
goal_node = (2, 5)

result_path = astar(maze, start_node, goal_node)

print("Resulting Path:", result_path)
```

Output:

```
Run    LA2Q4 ×
C:\Users\SCM\PycharmProjects\RLabAssignments\venv\Scripts\python.exe C:\Users\SCM\PycharmProjects\RLabAss
Resulting Path: [(5, 0), (4, 0), (3, 0), (2, 0), (1, 0), (1, 1), (1, 2), (1, 3), (2, 3), (2, 4), (2, 5)]

Process finished with exit code 0
```

Question No. 5:

Imagine going from Arad to Bucharest in the following map. Your goal is to minimize the distance mentioned in the map during your travel. Implement a uniform cost search to find the corresponding path.

Answer:

Code:

```
graph = {

    'Arad': {'Zerind': 75, 'Timisoara': 118, 'Sibiu': 140},

    'Zerind': {'Arad': 75, 'Oradea': 71},

    'Oradea': {'Zerind': 71, 'Sibiu': 151},

    'Timisoara': {'Arad': 118, 'Lugoj': 111},

    'Sibiu': {'Arad': 140, 'Oradea': 151, 'Fagaras': 99, 'Rimnicu Vilcea': 80},

    'Fagaras': {'Sibiu': 99, 'Bucharest': 211},

    'Lugoj': {'Timisoara': 111, 'Mehadia': 70},

    'Rimnicu Vilcea': {'Sibiu': 80, 'Pitesti': 97, 'Craiova': 146},

    'Mehadia': {'Lugoj': 70, 'Drobeta': 75},

    'Drobeta': {'Mehadia': 75, 'Craiova': 120},

    'Pitesti': {'Rimnicu Vilcea': 97, 'Bucharest': 101},

    'Craiova': {'Drobeta': 120, 'Rimnicu Vilcea': 146, 'Pitesti': 138},

    'Bucharest': {'Fagaras': 211, 'Pitesti': 101, 'Giurgiu': 90, 'Urziceni': 85},

    'Giurgiu': {'Bucharest': 90},

    'Urziceni': {'Bucharest': 85, 'Hirsova': 98, 'Vaslui': 142},

    'Hirsova': {'Urziceni': 98, 'Eforie': 86},

    'Eforie': {'Hirsova': 86},
```

```python
    'Vaslui': {'Urziceni': 142, 'Iasi': 92},
    'Iasi': {'Vaslui': 92, 'Neamt': 87},
    'Neamt': {'Iasi': 87}
}

import heapq

def uniform_cost_search(graph, start, goal):
    frontier = [(0, start)]
    explored = set()
    parent = {start: None}
    while frontier:
        (cost, current_node) = heapq.heappop(frontier)
        if current_node in explored:
            continue
        explored.add(current_node)
        if current_node == goal:
            path = []
            while current_node != start:
                path.append(current_node)
                current_node = parent[current_node]
            path.append(start)
            path.reverse()
            return (cost, path)
        for neighbor, neighbor_cost in graph[current_node].items():
            if neighbor not in explored:
                heapq.heappush(frontier, (cost + neighbor_cost, neighbor))
                parent[neighbor] = current_node
```

```
    return (-1, [])


start = 'Arad'

goal = 'Bucharest'

(cost, path) = uniform_cost_search(graph, start, goal)

if cost == -1:

    print(f"There is no path from {start} to {goal}")

else:

    print(f"The cost of the path from {start} to {goal} is {cost}")

    print(f"The path is: {' -> '.join(path)}")
```

Output:

Run LA2Q5

```
C:\Users\SCM\PycharmProjects\RLabAssignments\venv\Scripts\python.exe
The cost of the path from Arad to Bucharest is 418
The path is: Arad -> Sibiu -> Rimnicu Vilcea -> Pitesti -> Bucharest

Process finished with exit code 0
```